Chapter 1

Measurements for Software Aging

Roberto Pietrantuono, Javier Alonso and Kalyan Vaidyanathan

Università degli Studi di Napoli Federico II, Italy
Amazon PrimeAir, Seattle, USA
Oracle Corporation, Redwood City, USA

Email: roberto.pietrantuono@unina.it, jjava@amazon.com, kal.vaidya@gmail.com

A considerable attention has been devoted to the analysis of software aging based on measurements from real systems. This approach foresees the adoption of to infer, from collected data, the presence of aging trends (aging detection) as well as to forecast the future evolution of such trends in order to determine the optimal rejuvenation time.

This chapter will target the main methods adopted for the analysis and detection of software aging phenomena based on measurements. It will cover the methods for trend detection, estimation, forecasting as well as data manipulation. These methods can be classified as threshold-based approaches, statistical approaches for time series analysis and machine learning approaches for aging state classification and failure prediction. Mathematical details of the discussed techniques will be described in Appendix A-2.

1. Introduction

Due to the fact that the software aging phenomenon is relatively easy to be observed in any complex enough system, a significant effort has been dedicated from the academia as well as the industry \(^1\) to the analysis of software aging indicators (i.e., hardware and software metrics). These indicators are used to determine the existence of the phenomenon and, once determined, try to estimate the remaining time to live of the system under the effect of software aging, with the aim of determining the optimal moment to trigger the software rejuvenation process.

Depending on the techniques applied on the measurements collected from the system to estimate the presence of software aging and the estimation of the time to failure, we can talk broadly between two major approaches: threshold-based approaches or statistical-based approaches.

In both approaches, they monitor different type of system metrics like memory usage, CPU usage, number of system requests, response time or file descriptors use. These metrics are used to determine which parts of the system are involved in the aging phenomena, determine the time until the system reaches an unacceptable
state, and how to estimate the optimal moment to apply the recovery process. Within this family we need to distinguish threshold-based, statistical and machine learning approaches, depending on the approach used. Even, both families are focused on monitoring, studying and proactively act on the metrics’ values, the way each approach treats these metrics implies substantial differences in terms of limitation and accuracy of the approach or prior knowledge of the system under study.

Threshold-based approaches are purely based on applying some level of prior knowledge of the system as well as the metric (or metrics) that shows the aging effect and determine a threshold. If an aging metric reaches the prefixed threshold value, the rejuvenation process is triggered. These techniques are usually based on some level of expert human knowledge or historical data.

The second group of techniques, called for simplification statistical based approaches, can be further categorized depending on the actual statistical techniques applied. We differentiate between statistical techniques and machine learning based approaches in order to differentiate the techniques based on data driven learning processes like Machine Learning from traditional statistical methods and tests.

Statistical approaches take benefit from techniques like trend detection or time series analysis to determine if the aging phenomena actually exist and to estimate the time to failure of the system, respectively.

Finally, machine learning approaches leverage different machine learning algorithm families to try to predict the system crash due to software aging phenomena based on the indicators generated by the system like memory usage, file descriptors, CPU usage, etc.

Due to the relevance and differences of each of the approaches briefly introduced, this chapter devotes one section to each of them. It can be argued that some statistical techniques can be also considered as part of machine learning family of approaches or that some approaches present hybrid solutions involving techniques from two or more groups of approaches.

This level of overlap can make hard to classify each individual technique. However, in the opinion of the authors of the chapter, this proposed classification represents the best way to understand the idiosyncrasy and pros and cons of each family. So, when the reader confronts any new (hybrid or not) solution would be in a good spot to analyze it and understand the implications of such solution, even if it involves techniques from more than one methods here presented.

The rest of the chapter is thus organized as follows: Section 2 is focused on the threshold-based approaches. Section 3 analyzes the statistical approaches and describes the major techniques used within this field. Section 4 describes the main machine learning approaches used to try to predict when the system is going to crash due to the aging phenomena. Section 5 tries to help the reader providing a comprehensive list of metrics that have been historically linked with the aging phenomena. The main goal of this section is to provide a good repository of well-
known of aging metrics. Finally, Section 6 concludes the chapter.

2. Threshold-based approaches

This kind of approaches define thresholds for some aging indicators, and the rejuvenation is triggered when the monitored indicators exceed such thresholds. For instance, indicators may refer to resource consumption. Difficulties arise in identifying the best indicators and the right thresholds for them, that are able to prevent actual failures and useless rejuvenation actions at the same time. An examples of this approach is in the work by Silva et al., which adopts thresholds on mean response time and on quality of service indicators. Authors propose a rejuvenation approach based on self-healing techniques, that exploits virtualization to optimize recovery. They implemented a rejuvenation framework, called VM-Rejuv, in which an Aging Detector module detecting aging conditions based on the mentioned thresholds. Another work by Silva et al. is a further example of threshold-based approach; the paper presents an analysis of software aging in a SOAP-based server, in which a dependability benchmarking study is conducted to evaluate some SOAP-RPC implementations, focusing in particular on Apache Axis, where they revealed the presence of aging by parameters monitoring.

Thresholds are also adopted to evaluate the aging effects in Apace Web server based on a controlled experiment. The memory consumed by Apache is observed together with three controllable workload parameters: page size, page type (dynamic or static) and request rate; authors adopted thresholds on the usage of virtual memory as aging indication.

Thresholds-based approaches can be combined with others. For instance, a threshold could be established for some parameters for which a statistical or machine learning approach are hard to apply (e.g., because of the characteristics of the observations such as high noise or variability), while time-series or machine learning approaches for other indicators more prone to be dealt with statistical tools. Such an approach has been applied on the Eucalyptus cloud computing platform: Araujo et al. combined the threshold-based approach (with a threshold on memory utilization) with time-series analysis, implementing a rejuvenation policy in Eucalyptus by using multiple thresholds and forecasting by time series analysis models. Time-series models adopted are: linear model, quadratic model, growth curve model, and S-curve trend model.

Thresholds on resources are also adopted by Avritzer et al., where threads and memory are monitored and actions are taken when they exceed some thresholds (e.g., garbage collection).

3. Statistical approaches

A widely used technique for the measurements-based approach is time series analysis. Variables potentially related to software aging, such as system’s resources
consumption or user-perceived performance indicators, are monitored periodically over time, and the resulting time series are analyzed to assess the presence of aging phenomena. Time series analysis for aging assessment copes with two main problems: 1) detecting the presence of a degradation trend, and, if such a degradation trend is present, 2) quantitatively estimating the extent of the degradation and its characteristics (e.g., seasonality in data). These are dealt with statistical trend detection and trend estimation techniques, respectively. Trend detection applies statistical hypothesis test to determine whether the values within a time series (i.e., a series of observations of a random variable) generally increase (or decrease) over a long period of time. The statistical test allows making conclusive statements about the presence or absence of software aging trends with a desired statistical confidence. Trend estimation allows quantitatively characterizing the long-term component of a time series.

The most used trend detection techniques in software aging research are, by far, the Mann-Kendall and Seasonal Kendall test. They are hypothesis tests (for non-periodic and periodic time series, respectively) to accept/reject the hypothesis of no trend in data. Being non-parametric tests, they do not make assumptions on the distribution of data and are more robust to outliers than parametric tests; on the other hand, they are generally conservative, namely some actual trend could be missed. Mann-Kendall (and Seasonal Kendall) test is usually applied in conjunction with the Sen’s (and Seasonal Sen’s) trend estimation procedure for trend estimation. The Sen’s procedure is also a non-parametric linear regression technique (also insensitive to outliers) that fits a linear model and computes the rate at which the samples increase over time by simply taking the slope as the median of all slopes between paired values.\textsuperscript{11,12} This combination (Mann-Kendall plus Sen’s estimator) is actually applied in a huge number of aging research studied, including.\textsuperscript{2,13–15}

One of the first measurement-based analyses applying the Mann-Kendall and Sen’s procedure was by Garg \textit{et al.}\textsuperscript{2} Until then, observations of software aging had only been mainly anecdotal. Instead of collecting metrics on failure events (which most research on software reliability rely on), the main idea behind this work was to periodically monitor and store data on the attributes of a software system that are indicative of the "health" of the system.

Using the SNMP (Simple Network Management Protocol) framework, a distributed resource monitoring tool was designed and developed. The goal was to monitor the health of UNIX (Solaris) workstations at the OS level. The objects (or metrics) that were monitored broadly fell into seven categories - host details (that are constant), timestamp, OS resources, process states, file system resources, network resources and I/O resources. The SNMP agent ran on each monitored workstation while the monitoring station ran the SNMP manager and acted as a central data repository. Eight workstations were monitored for approximately two months with the frequency of data collection set to fifteen minutes.

Some of the questions that the authors attempted to answer were (a) is aging
present (b) how do the metrics behave over time (c) can observed failures be related to specific metric values and (d) can software aging be quantified? Linear and periodic dependency analysis (autocorrelation and harmonic analysis) and trend detection and estimation techniques (seasonal Kendall test) were used to answer these questions. Daily and weekly dependencies were observed in many of the monitored variables. Smoothing techniques (non-parametric local weighted regression) and trend detection (seasonal Kendall test) were applied to the time-series data to detect global trends that pointed to evidence of software aging.

One of the major contributions of this paper was aging quantification and estimations of time to failure due to aging. The quantification aspect was addressed by calculating a slope metric using Sen’s non-parametric method for each metric and the corresponding time to failure estimation was performed using a simple linear projection, given the minimum/maximum of the metric of interest.

The above work was extended by Vaidyanathan and Trivedi, where a measurement-based model was proposed to estimate resource exhaustion times based on time as well as system workload state. The same setup as was used to monitor and collect UNIX data. Statistical clustering techniques were employed to identify different workload states and a semi-Markov state space model was developed. This approach can be also understood as a Machine Learning approach. For each of the monitored metrics of interest, a reward function was computed based on the exhaustion rate of these metrics (resources) at each of the identified workload states. The expected accumulated reward was used an indicator of the resource usage trend and estimates of failure times were computing using this. This work clearly demonstrated the relation between system workload and resource exhaustion and supports prior studies that showed workload does affect system reliability/availability. Not only was the system workload dynamics captured in the model but also the effect of workload on resource usage was quantified by means of reward rates or slopes in the model. In doing this, the software aging phenomenon was validated with respect to resource exhaustion. This workload-based model was further expanded in by feeding the results to a higher level availability model that accounts for failure followed by reactive as well as proactive recovery. Optimal rejuvenation schedules are derived that minimize downtime cost and maximize availability.

Grottke et al analyzed performance degradation in the Apache Web Server by sampling web server’s response time to predefined HTTP requests at fixed intervals, using a similar procedure adopted by Garg et al., also analyzing seasonal patterns in which trends analysis accounts for the possible presence of seasonal variation in data.

More recently, Zheng et al proposed an alternative to the conventional (Seasonal) Mann-Kendall test and Sen’s procedure. The authors raise some criticalities to the Mann-Kendall test and Sen’s trend estimator (e.g., suitability for linear trends only, sensitivity to noise and computational complexity) and propose to use a modified
version of the Cox-Stuart test for trend detection and the iterative Hodrick-Prescott Filter for (linear and non-linear) trend estimation. The Cox-Stuart test is based on the sign test between pairs of a series of observations (a modified version is proposed for periodic time series). The Hodrick-Prescott filter does not presume a structure of the trend (e.g., linear trend); it solves a “penalized” spline model, fitting the raw time series to be a more smoothened representation, by using a smooth parameter that penalizes the variability of the trend component, thus controlling the trade-off between the goodness of fit and smoothness (the larger its value, the smoother the trend component). It is insensitive to periodicity too.

Together with the conventional Mann-Kendall for trend detection, Li et al.\textsuperscript{17} used time-series ARMA/ARX models for trend estimation on the Apache Web Server, in order to estimate the resource exhaustion. Compared with the linear regression and extended linear regression models, ARX model incurs higher initial overhead, but once it is established, it can be used for prediction for a long period without re-estimating the parameters in the model.

ARIMA (Autoregressive Integrated Moving Average) and Holt-Winters (Triple Exponential Smoothing) models have been used by Magalhaes and Silva,\textsuperscript{18} wherein authors developed a framework for detection of performance anomalies caused by aging, which is targeted to web and component-based applications. In particular, the framework monitors application/system parameters, used to determine the correlation between the application response time and the input workload, in turn used to train machine learning (ML) algorithms. At run-time, parameters collected by monitoring are estimated by ARIMA and Holt-Winters algorithms, and the estimations classified by the trained ML algorithms to determine if the application may incur in some performance anomaly.

Time-series analysis has been also adopted to study the relationship of the software aging and workload in complex systems, including the Linux Kernel code,\textsuperscript{19} and the Java Virtual Machine.\textsuperscript{20} In both cases, the analysis of workload parameters is used to provide indications on potential sources of software aging, by highlighting the subsystems whose parameters are correlated to the experienced aging trends. In the latter, parametric trend detection was used for aging detection, specifically the conventional \textit{t-student} test. Then, Principal Component Analysis (PCA) followed by multiple linear regression are also adopted, in order to remove first-order correlation among predictors and then to provide linear estimates of aging trends, so as to reduce the problem of multicollinearity. Clustering is also preliminarily exploited to separate out different workload states possibly exhibiting different trends. In the work by Bovenzi et al.\textsuperscript{21} the authors adopt a Design of Experiment (DoE) technique in order to assess the presence of aging under different workload configurations in several applications (the James mail server, a middleware for air traffic control system, the Apache web server, and the MySQL DBMS): series of controlled long-running experiments are planned and executed, and the resulting time series are analyzed by conventional Mann-Kendall and Sen’s procedure followed by the
Analysis of Variance (ANOVA) to assess the impact of various workload factors on software aging. The impact of concurrency bugs on aging is also assessed by this technique.

Non-linear time-series analyses are used by Araujo et al.\textsuperscript{9} Four different time-series models have been used to schedule software rejuvenation properly: the linear model, the quadratic model, the exponential growth model and the model of the Pearl-Reed logistic. They have been adopted for predicting memory consumption trends on the Eucalyptus cloud computing framework. One further paper considering nonlinear models is by Jia et al.\textsuperscript{22} in which aging is studied in Apache by constructing a dynamic model to describe the software aging process following the method of nonlinear dynamic inversion. Software aging process is shown to be nonlinear and chaotic.

In the best practice guide by Hoffmann et al.\textsuperscript{23} multivariate nonlinear models (support vector machines, radial, and universal basis functions) have been compared with multivariate linear models. The former ones have shown better performance than linear models in the benchmarking case studies.

Finally, Chen et al.\textsuperscript{24} propose a new aging metric based on Multidimensional Multi-scale Entropy (MMSE) for trend analysis and aging-related failure prediction. PCA is leveraged to select variables from an initial set of multiple aging indicators; on normalized time series of the selected indicators, the MMSE metric is computed, where a greater entropy is demonstrated to relate to a more severe aging phenomenon.

4. Machine learning approaches

As stated in previous section, there are several papers using time series approaches as first step in the area of Machine Learning. If we consider software aging phenomena as a result of two factors (the system resource presenting the aging bug and time), time series approaches could be considered ideal. However, these approaches are drastically limited by the forehand knowledge about the resource/metric that is affected by the aging phenomena or at least know the affected performance metric (e.g., response time, throughput). This represents an increasing barrier to adopt time series approaches with the increasing complexity of the systems and the interdependency between resources at different system layers.

In this scenario Machine Learning approaches come to place. Machine Learning approaches are a more sophisticated form of data analysis, which adopt machine learning algorithms (e.g., classifiers and regressors) to identify trends (e.g., estimating time to failure) and classify a system state as robust or failure-prone.

Pattern recognition methods have been used to predict software aging in shared memory pool latch contention in large OLTP servers.\textsuperscript{25} The approach applies non-linear, non-parametric regression to a large set of system variables, and analyze the residual error between the predicted and the actual system values using a sequential
probability ratio test, in order to predict the onset of software aging effects. Results showed that these methods allowed to detect significant deviations from “standard” behavior with a 2 hours early warning.

We subdivide the machine learning approaches based on machine learning algorithms used into two major families: regression-approaches and classification approaches.

The former approach is focused on trying to predict the time to exhaustion (TTE) of the system resource or resources due to aging phenomena. The later approach is focused on determining the state of the system (i.e., stable vs non-stable state).

There are several published papers within the domain of predicting the time to crash due to software aging. For example, in, the authors compare different regression algorithm families (i.e., regression trees, linear regression and hybrids). The authors were focused on comparing these algorithms within different scenarios and multiple aging phenomena involved. The conclusion obtained by the authors was that the aging phenomena was better modeled by a hybrid model (i.e., MP5) between Decision Trees and Linear regression, since the aging phenomena caused by resource exhaustion due to bugs in the software (i.e., memory leaks or threads unreleased after use) can be modeled by linear piecewise models capturing different aging slopes or speeds. Further three different machine learning algorithms (namely, naive Bayes classifier, decision trees and a neural network model) have been also used, in combination with time-series models, in order to predict aging in web applications.

They automatically built the models relating several system variables (e.g., number of connections and throughput) to aging trends, based on the observation that software aging trends can be approximated using a piecewise linear model. The models were trained using data samples collected in preliminary experiments, and were used to predict the TTE of system resources under conditions different than the ones observed during the training phase.

An example of the application of machine learning classifier algorithms is. Alonso et al. compare a large set of families (i.e., random forest, decision trees, LDA/QDA, Naive Bayes, Support Vector Machines and K-nearest neighbors) to predict state in the context of a three-tier J2EE system.

In, the authors compare four classification algorithms: J48 (a decision tree based algorithm), Navie Bayes, Support Vector Machines (i.e., SMO) and ZeroR (The 0/R classifier). The authors compare these four algorithms under software aging scenarios caused by only one metric under constant software aging injection rate. The experimental evaluation suggested that all these four main families of classification techniques perform similarly, even SMO presents better overall results.

An approach not exactly falling in this area but still related is to use reinforcement learning for aging estimation. Eto, Dohi and Ma used reinforcement learning to estimate the optimal rejuvenation schedule adaptively, i.e., by consider-
ing runtime data to update the estimation. Thus, their estimation technique does not require the complete knowledge on system failure (degradation) time distribution in the operational phase, even if the underlying state transition of software is governed by models, i.e., by CTMCs or SMPs. This is also an example of a hybrid approach, wherein a measurements-based approach is used in conjunction with a model-based one.30

5. Relevant software aging metrics: beyond memory leaks

Aging indicators are an important area of study for measurements-based approaches. The correct identification of metrics better representing the aging of the system is of paramount importance to have a clear view on the system health’s state. Aging indicators can refer to resource usage and to performance as well. The following classes of aging indicators are commonly used in measurements-based aging analysis techniques:

- **Memory consumption**: metrics related to memory consumption are are by far the most commonly adopted ones for monitoring resource depletion. Empirical evidence showed that free memory exhibits the shortest Time to Exhaustion (TTE) among system resources2 and that memory management defects are a significant cause of aging failures31 and of failures in general.32 Therefore, many measurements-based studies analyze aging phenomena affecting free memory, by measuring the amount of free physical memory and swap space,3,13,33 applying often time series and statistical models to these variables.

- **Performance degradation**: Measurements-based techniques often refer to performance degradation in software systems affected by aging, which is the other effect (together with resource depletion) that aging is expected to cause. Performance degradation is indeed related to memory usage: for instance, the consumption of physical memory increases the time required by memory allocation procedures and garbage collection mechanisms, since their computational complexity is a function of the amount of memory areas that have been allocated.20,34,35 But performance degradation can be also due to other, more complex and hard-to-reproduce, aging causes such as fragmentation or concurrency management.21,36 Common indicators are response time, latency, throughput, transactions rate or, less commonly, number of SLA violations. In the presence of this kind of phenomena, software rejuvenation can be triggered when the quality of service (e.g., in terms of response time or throughput) is below a given threshold.

- **Other resource consumption**: Software aging can impact on several kind of other resources. Besides memory-related resources (e.g., physical memory, virtual memory, swap space, cache memory), analyzed papers deal with the these type of resources:
filesystem-related resources, such as stream descriptors and file handles\textsuperscript{2,37,38}

storage, whose space may be consumed by bad management\textsuperscript{39}

network-related resources, such as socket descriptors\textsuperscript{37}

concurrency-related resources, such as locks, threads and processes\textsuperscript{2,21,38}

application-specific resources, such as DBMS shared pool latches\textsuperscript{25} and OSGi references.\textsuperscript{40}

In many cases, the approach is not constrained to a specific resource, but is focused on detecting incorrect API usage and incorrect exception handlers that may cause a resource leakage. For instance, in the work by Zhang et al., the authors present an approach that dynamically mines resource usage patterns by monitoring API calls, and provides an experimental evaluation on open source programs based on the Java I/O and concurrency APIs.\textsuperscript{38}

A frequent kind of resource leak in Java programs is represented by sockets and file handles, due to faulty exception handlers that do not release these resources.\textsuperscript{37,38} Other resources can also be affected by software aging depending on the kind of system, such as free disk space in DBMS.\textsuperscript{39} In some cases, a wider set of resources can be analyzed. The mentioned example by Garg et al. applies time series analysis on a network of UNIX workstations monitored on several resources (related to virtual memory, the OS kernel, the filesystem, the disk, and the network), noticing a statistically significant trend in the process table size and in the file table size (although their TTE was lower than the TTE of free memory). Other resources include CPU utilization, power consumption, number of threads/processes. In cloud systems, indicators can be measured at any layer of the virtualization technology stack, e.g., at application or OS level within the VM layer, at VM layer to probe the state and resource consumption of the VMs (e.g., for load balancing, scaling, VM migration and rejuvenation decisions), as well as at VMM layer. Besides system resources, other indicators of interest are related to VMs, for instance, the number of VM new allocations/releases, the time to start/stop VMs or to the time to migration.

Further indicators are being considered more recently. A recent field in which software rejuvenation is being studied is related to security attacks, that is, attempts of malicious users to access unauthorized resources, to cause their gradual leakage or to make the system unavailable. In fact, security attacks may take place and gradually compromise a system over a long period of time (e.g., password theft through brute-force guessing, or flood attacks that trigger software aging phenomena), which can be mitigated by periodically rejuvenating a system, such as by changing cryptographic keys, by restarting compromised processes, and by randomizing the location of data and instructions in memory.\textsuperscript{41–46} Currently, the aging rate has to be assumed
at design time\textsuperscript{41,47} or should be based on imperfect attack/intrusion detectors that could raise false alarms and miss attacks.\textsuperscript{48,49}

Other aging indicators may be aimed at capturing effects like accumulation of numerical errors\textsuperscript{50} and memory fragmentation.\textsuperscript{50,51} This kind of aging effects are not necessarily caused by bugs in the software, but are related to the nature of floating-point arithmetic and memory allocation algorithms, respectively.

6. Conclusions

This chapter described the main approaches to deal with software aging based on measurements. Unlike model-based approaches, measurements-based techniques rely on observations from real systems, with the goal of i) detecting whether the system is in a failure-prone state due to software aging, of forecasting the time-to-aging-failure, and of planning software rejuvenation accordingly.

The advantage of such solutions lie in the possibility to gather accurate and detailed information about the aging state of the system under analysis, and there is no need to make assumptions on model parameters, because real data are available.

However, while the techniques described in this chapter can be applied to any system for which monitoring is possible, it is also true that the gathered observations from one system are hardly generalizable. A good example of this are the machine learning models trained with such data. They may work appropriately with the same system from the data is coming but it is unclear how much they can be generalized across similar systems.

Moreover, at design time there is no data available to determine potential sources of possible aging behaviors – the real system is needed for data to be collected. This requires a tuning of the technique used based on the system to be analyzed, which considers the indicators that are more likely related to the aging phenomena of that specific system – hence more explanatory of aging behaviors – and that can be monitored with acceptable overhead.

Due to the underlying limitations of the measurement based approaches, makes natural the combination of measurements-based approaches with model-based. Real data can be used to estimate the parameters for the models. So, combining both approaches increases our capabilities to study the effect of the software aging on a given system in a much more detail that simple observations.

In the future, we expect an increasing impact of measurements-based aging analysis technique in the SAR literature. We expect that Machine Learning techniques increase their presence in the literature because in some sense they represent a great hybrid between model based and measurement based approaches. Indeed, as systems become more and more complex, much more indicators – and suitable combination thereof – need to be analyzed with respect to their relation with software aging.

On the one hand, more indicators in increasingly complex systems can provide
many useful hints to detect software aging phenomena. On the other hand, it becomes increasingly difficult to distinguish aging phenomena from expected fluctuations of the system performance and of resource usage patterns, because of high inter-relations among multiple parameters. All these call for more complex statistical techniques able of analyzing the cause-effect relations (and not only correlations) among multiple indicators and of spotting those critical patterns relevant for aging identification in complex systems. This is a challenge as much difficult as of great importance for the increasing impact of aging problems in today’s systems.

References


Measurements for Software Aging


